Funcionamento e Soluções (Pipeline)

APRESENTAÇÃO

Ao adquirir ou utilizar um computador, o requisito básico imposto pelo consumidor ou usuário é a *performance*. Nenhum usuário se sente confortável e satisfeito ao utilizar um computador no qual, para executar qualquer ação, seja necessário aguardar um tempo de processamento significativo. A *performance* dos computadores é influenciada por diversos fatores, entre eles: tipo de processador e tipo e tamanho das memórias. Pensando em melhorar a *performance* dos computadores, foi agregada a eles a tecnologia de *pipeline*. Essa tecnologia permite que o processador fragmente uma instrução grande, a qual demoraria um tempo significativo de processamento, em partes menores, para as quais são dedicadas partes separadas do processador para executar. Todas as partes do processador são utilizadas, não deixando *hardware* ocioso, trabalhando paralelamente de forma independente.

O conceito de *pipeline* é aplicado não apenas na informática, no que se refere a uma tecnologia de processamento, mas também em diversas outras áreas. Um exemplo de *pipeline* bem comum, e não relacionado à informática, são os atendimentos bancários, nos quais várias pessoas atendem clientes simultaneamente, cada uma fazendo um tipo de serviço e, assim, agilizando os atendimentos.

Diversos são os tipos de *pipeline*, sendo o de instruções o mais importante. O *pipeline* de instruções é aplicado em diversas situações, quando o computador precisa executar uma instrução, como, por exemplo, ler e copiar um arquivo pesado.

Nesta Unidade de Aprendizagem, você aprenderá sobre uma das tecnologias utilizadas para otimização do processamento de um computador e ganho de tempo na execução de atividades complexas.

Bons estudos.

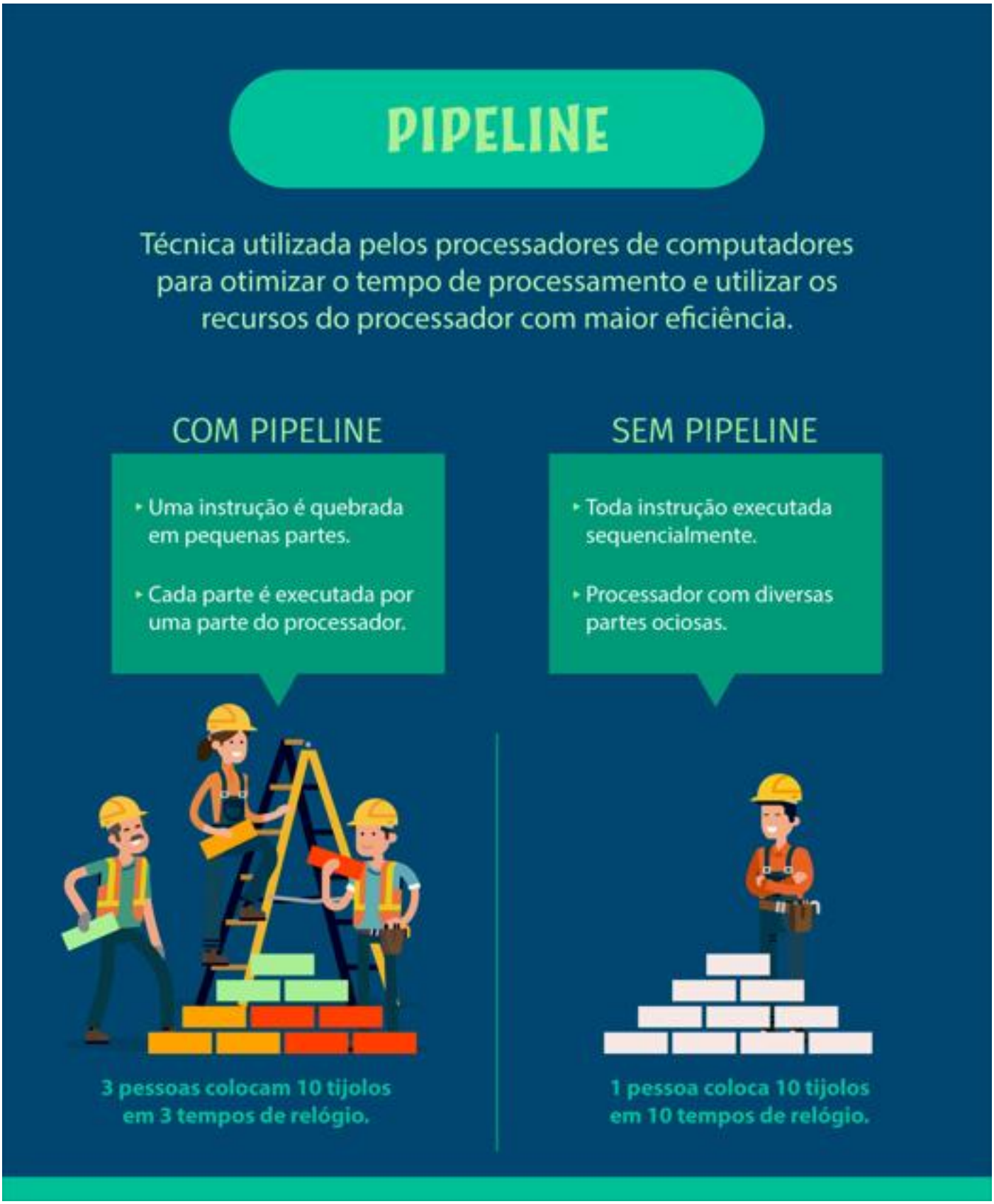
Ao final desta Unidade de Aprendizagem, você deve apresentar os seguintes aprendizados:

- Descrever o conceito de *pipeline*.
- Identificar as características dos *pipelines* de intruções.
- Reconhecer os tipos de *pipelines*.

INFOGRÁFICO

Os processadores com *pipeline* dividem as tarefas que precisam executar em partes menores, sendo que cada parte do processador irá se dedicar a uma parte da atividade. Isso otimiza o tempo de processamento, fazendo com que os recursos do processador sejam utilizados com maior eficiência.

O Infográfico a seguir apresenta uma analogia desse conceito com a construção de um muro sendo feita por um ou por três profissionais.



CONTEÚDO DO LIVRO

Pipeline é uma tecnologia utilizada pelos processadores de computadores, a qual permite dividir

uma tarefa ou instrução em várias partes, ficando cada parte do processador responsável por uma parte da atividade. Dessa forma, todas as partes podem ser executadas simultaneamente, resultando em ganho de tempo.

No capítulo Arquitetura e organização de computadores, do livro *Funcionamento e soluções (pipeline)*, base teórica para desta Unidade de Aprendizagem, você vai aprofundar os seus conhecimentos sobre *pipeline*.

Boa leitura!

ARQUITETURA E ORGANIZAÇÃO DE COMPUTADORES

Aline Zanin

Funcionamento e soluções (*pipeline*)

Objetivos de aprendizagem

Ao final deste texto, você deve apresentar os seguintes aprendizados:

- Descrever o conceito de *pipeline*.
- Identificar as características dos *pipelines* de instruções.
- Reconhecer os tipos de *pipeline*.

Introdução

Podemos dizer analogamente que o processador — unidade central de processamento (UCP, ou CPU, do inglês *central processing unit*) — é o coração do computador. Nenhuma atividade é executada sem passar por ele. Em alguns casos, as instruções que precisam ser processadas pela CPU são complexas e demandam tempo. Para reduzir esse tempo, são utilizadas estratégias de paralelismo.

Existem dois tipos principais de paralelismo. O paralelismo em nível de *hardware é* alcançado com a replicação do número de unidades de processador, de modo que eles trabalhem em paralelo. Já o paralelismo em nível de instruções, conhecido como ILP (do inglês *instruction level parallelism*), trabalha com a organização das unidades do processador, para que não fiquem ociosas.

Há duas formas principais de implementar o ILP: uma delas é por meio do *pipeline*, e a outra é por meio de processadores superescalares. *Pipeline* é um conceito empregado pelos processadores no processamento de tarefas. Com ele, o processador paraleliza as atividades que precisam ser executadas, fazendo diversas atividades ao mesmo tempo, otimizando o trabalho e, reduzindo o tempo demandado para essas atividades.

Assim, neste capítulo, você vai estudar sobre o *pipeline*, verificando os tipos existentes e as características principais de um *pipeline* de instruções.

O que é *pipeline*

A busca por otimizar o processamento do computador, paralelizando a execução das instruções, não é recente. Desde o IBM Stretch (1959), os computadores já tinham a capacidade de buscar instruções em memória e armazená-las em um *buffer*, o que permitia que essas instruções fossem acessadas antecipadamente, isto é, antes da conclusão da leitura da memória do computador. Esse processo de busca antecipada, na verdade, paralelizava o processo, porque o dividia em duas partes, sendo elas a busca e a execução propriamente dita das instruções. O conceito de ***pipeline*** amplia essa estratégia, porque permite dividir uma instrução em diversas partes e dedicar uma parte do *hardware* para a execução de cada divisão da instrução, conforme Tanenbaum (2007).

O conceito de *pipeline* pode ser aplicado em situações cotidianas, e não apenas no contexto dos processadores de computador. Um exemplo claro é uma linha de produção, seja de carros e hambúrgueres. Imagine o seguinte cenário: em uma lanchonete, há cerca de 30 clientes na fila, e apenas um funcionário trabalhando. Esse funcionário executa sequencialmente as ações de: assar o pão, fritar o hambúrguer, cortar o pão, passar o molho no pão, fritar as batatas e, finalmente, montar o sanduíche.

É notório que esse processo seria otimizado caso essas ações fossem executadas paralelamente: enquanto o pão está assando, o hambúrguer pode ser assado; enquanto isso as batatas são preparadas; o pão é cortado e recebe o molho O mesmo acontece no computador: se o processador executar todas as tarefas sequencialmente, precisará de mais tempo. Se executá-las em paralelo, o tempo necessário será menor, aponta Tanenbaum (2007).

Todo processo que faz uso de *pipeline*, seja ele um *software* ou não, considera duas premissas básicas:

1. O processo é dividido em etapas independentes umas das outras.
2. Um novo produto inicia sua produção antes que o produto anterior tenha sido concluído.

O *pipeline* não reduz o tempo gasto para completar cada instrução individualmente — o ganho de tempo está no processamento simultâneo de diversas instruções, e não na velocidade de processamento individual de cada instrução. Esse processo de executar diversas instruções simultaneamente é chamado de paralelismo de instruções e tem por objetivo aumentar o número de execuções realizadas a cada ciclo do relógio.

Fique atento

Existe uma metodologia ágil de desenvolvimento de *software* que prega que tudo o que é bom deve ser explorado ao extremo (*extreme programming*, XP). No caso da arquitetura de processadores, se um *pipeline* é bom, por que não colocar dois, três ou mesmo quatro? Quando se replicam os *pipelines*, constrói-se o que chamamos de **arquitetura superescalar**.

Pipeline de instruções

As ações de processamento executadas pelo processador de um computador são conhecidas como instruções. Na seção anterior, vimos que um *pipeline* existe para paralelizar um trabalho, fazendo com que a sua execução deste em pequenas partes seja mais eficiente e mais rápida.

Um ***pipeline*** de instruções nada mais é do que a aplicação desse conceito nas instruções recebidas pelo processador. Ou seja, ao receber uma determinada instrução para ser processada, o processador vai dividi-la em etapas conforme o seu tipo e tamanho, para que sejam dedicadas partes do *hardware* para cada pedaço da instrução.

Saiba mais

Opcode (do inglês *operation code*) ou código de operação é um código fundamental para o funcionamento dos processadores, que é gerado em todas as operações executadas. Esse código é um intermédio entre a linguagem humana e a linguagem binário interpretada pelo processador.

Antes de tratarmos do paralelismo de instruções, vamos ver um pouco do ciclo natural que uma instrução tem em seu processamento. A execução de uma instrução é naturalmente dividida nas etapas de busca-decodifica e executa. Esse ciclo acontece da seguinte maneira: a CPU busca uma instrução, transferindo-a da memória principal para o registrador de instruções; poste-

riormente, essa informação é decodificada, ou seja, é determinado o *opcode*, e são carregados os dados necessários para a execução da instrução; por fim, a instrução é executada, com todas as suas operações.

Considere uma CPU com capacidade de execução de *pipeline*. Essa CPU pode receber uma instrução e quebrar esse processo em alguns "minipassos"; por exemplo:

1. buscar instrução;
2. decodificar *opcode*;
3. calcular endereço efetivo dos operandos;
4. carregar operandos;
5. executar instrução;
6. armazenar resultado.

Todos os estágios de um *pipeline* são conectados por uma estrutura chamada de duto, de forma que a instrução possa entrar em um lado do *pipeline* e sair concluída no outro lado. Na parte superior da Figura 1, temos um *pipeline* com cinco unidades; cada unidade está executando uma parte de uma tarefa relacionada à leitura e ao armazenamento de dados no computador. Na parte inferior da figura, podemos ver como esse *pipeline* funciona em função do tempo: a cada ciclo do relógio, um novo estágio é executado em paralelo com o estágio anterior, que já estava em execução. É possível que uma nova tarefa se inicie sempre sem a conclusão da tarefa anterior e que diversas tarefas sejam executadas ao mesmo tempo.

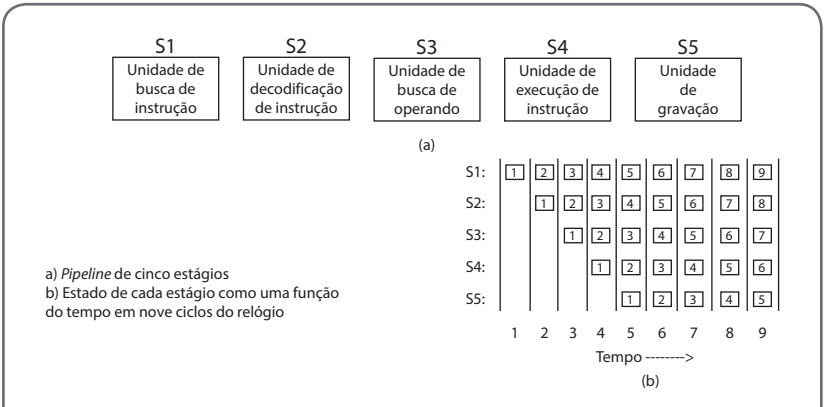


Figura 1. Ciclo de *pipeline*.

Fonte: Tanenbaum (2007, p. 35).

Um *pipeline* de instruções pode gerar dois problemas de conflito: conflito de recursos e dependência de dados. O conflito de recursos acontece quando duas ou mais instruções precisam acessar um recurso do computador simultaneamente. Por exemplo: quando uma instrução está armazenando um valor na memória, e, ao mesmo tempo, outra instrução está buscando valores na memória, ocorre um conflito de recursos, porque ambas necessitam acessar a memória. Geralmente, isso é resolvido ao se permitir que a instrução em execução prossiga, forçando a nova instrução a esperar, para depois cumprir o seu papel. Já a dependência de dados acontece quando uma instrução depende dos dados resultantes de outra instrução para poder ser executada ou para concluir o seu objetivo. Esse conflito, em geral, é resolvido com um tratamento de *hardware*, que vai identificar um possível conflito e retardar a execução da instrução dependente.

Link

Na animação disponível no link abaixo, você encontra uma explicação bastante didática sobre *pipelines* e arquiteturas superescalares, podendo ver de forma prática a interferência dos *pipelines* na performance de processamento do computador. Para assistir, acesse o *link* ou código a seguir.

https://goo.gl/aCq3Pf

Tipos de *pipeline*

São diversos os tipos de *pipeline* que podem ser implementados pelo processador para executar tarefas. Todos eles têm o mesmo objetivo: proporcionar maior agilidade e eficiência, reduzindo a ociosidade do processador. A seguir, vamos conhecer os principais tipos de *pipeline* e discutir suas características.

Fique atento

Para que os *pipelines* possam ser executados pela CPU, os *softwares* precisam estar programados para serem executados de forma paralela, explorando toda a capacidade do computador. Um exemplo de tecnologia utilizada para a construção de programas a serem executados utilizando paralelismo é o *Message Passing Interface* (MPI), um padrão para comunicação de dados em computação paralela.

Os *pipelines* se dividem da seguinte forma:

- *Pipelines* classificados por tipos de processamento:
  - *Pipeline* de instruções — é o *pipeline* descrito na seção anterior; refere-se ao processamento de instruções pelo processador do computador, podendo ser qualquer instrução; por exemplo, copiar um arquivo.
  - *Pipelines* aritméticos — são utilizados para a execução de operações aritméticas complexas — como multiplicação inteira, operações em ponto flutuante e operações vetoriais —, que podem ser divididas em etapas de execução menores; por exemplo, no caso de algoritmos de ordenação de vetores.
- *Pipelines* de controle de fluxo de dados:
  - Assíncrono — nesse método, os *pipelines* se comunicam por meio de sinais de *handshaking*, que são utilizados para indicar a disponibi-lidade de dados do estágio atual para o próximo estágio (RDY) e para indicar a liberação dos dados do estágio atual para o estágio anterior (ACK).
  - Síncrono — nesse estágio, os*pipelines* são interconectados por registradores (*latches*), que armazenam os dados intermediários durante a transferência entre estágios do *pipeline*. Nesse tipo de *pipeline*, toda transferência é controlada por um sinal de relógio, e o estágio com operação mais lenta determina a taxa de operação do *pipeline*.
- *Pipeline* de funcionalidade:
  - Unifuncional e multifuncional — *pipelines* unifuncionais são dedicados à execução de uma única operação, e *pipelines* multifuncionais podem executar mais de uma operação.
- *Pipeline* de estrutura:

- Linear — é um *pipeline* em que a execução dos ciclos segue um padrão sequencial, sem alterações de percurso.
- Não linear — é um *pipeline* em que o fluxo de execução dos estágios do *pipeline* sofre alterações de rota com frequência, podendo, inclusive, estabelecer novos ciclos.

Exemplo

Para entender melhor o conceito de *pipeline*, vamos pensar sob a perspectiva do *software*. Alguns *software* necessitam de grande capacidade computacional para atingir o seu objetivo, sendo que, algumas vezes, precisar de um grande tempo de espera para chegar ao resultado final. Existem diversos *software* desse tipo nas áreas de biologia e medicina, por exemplo.

Um exemplo de utilização de paralelismo são os algoritmos de ordenação de vetores. Imagine um vetor de 1 milhão de posições — se o processador efetuar a ordenação desse vetor de forma sequencial, vai precisar de um tempo significativo. Entretanto, se o programa quebrar o vetor em partes, e cada parte for executada separadamente por uma parte do processador, o tempo de ordenação do vetor será reduzido. Isso ocorre mesmo considerando-se que serão necessárias trocas de mensagens entre os estágios do *pipeline*, para garantir a ordenação total.

Referências

BRITO, A. V. *Introdução a arquitetura de computadores:* aumentando o desempenho com pipeline. [2018]. Disponível em: <http://producao.virtual.ufpb.br/books/edusantana/introducao-a-arquitetura-de-computadores-livro/livro/livro.chunked/ch02s07.html>. Acesso em: 18 dez. 2018.

CONHECENDO o Blog Opcode. [2018]. Disponível em: <http://opcode.com.br/opcode-blog/>. Acesso em: 18 dez. 2018.

MANACERO JR., A. *UCP e pipelines*. [2018]. Disponível em: <https://www.dcce.ibilce.unesp.br/~aleardo/cursos/arqcomp/pipelines.pdf>. Acesso em: 18 dez. 2018.

TANENBAUN, A. S. *Organização estruturada de computadores*. 5. ed. São Paulo: Pearson, 2007.

Encerra aqui o trecho do livro disponibilizado para esta Unidade de Aprendizagem. Na Biblioteca Virtual da Instituição, você encontra a obra na íntegra.

Conteúdo:

DICA DO PROFESSOR

O *pipeline* proporciona um grande ganho de *performance* para os computadores. É importante conhecer a importância do *pipeline* e seu funcionamento para saber adequar eficiência e o custo no momento da compra de um computador, bem como para, ao criar seus *softwares*, poder considerar a utilização de *pipeline*.

Nesta Dica do Professor, você vai conhecer um dos tipos mais importantes de *pipeline*, o *pipeline* de instruções.

Conteúdo interativo disponível na plataforma de ensino!

EXERCÍCIOS

1) **Considere uma CPU com capacidade de execução de pipeline. Ela pode receber uma instrução e quebrar esse processo em alguns minipassos. Assinale a alternativa que indica corretamente a ordem de execução desses minipassos.**

A) Carregar operandos, calcular endereço efetivo dos operandos, buscar instrução, armazenar resultado, decodificar opcode e executar instrução.

B) Buscar instrução, decodificar opcode, calcular endereço efetivo dos operandos, carregar operandos, executar instrução e armazenar resultado.

C) Buscar instrução, executar instrução, decodificar opcode, carregar operandos, calcular endereço efetivo dos operandos, e armazenar resultado.

D) Buscar instrução, calcular endereço efetivo dos operandos, carregar operandos, decodificar opcode, executar instrução. armazenar resultado.

E) Carregar operandos, calcular endereço efetivo dos operandos, decodificar opcode, buscar

instrução, executar instrução e armazenar resultado.

2) **O processador é a parte principal do computador, cabendo a ele a realização de funções como leitura e armazenamento de arquivos. O desempenho do processador interfere diretamente na *performance* do computador, e é por isso que os processadores têm técnicas para melhorar sua eficiência. Uma dessas técnicas é o *pipeline*.**

A respeito do *pipeline*, assinale a alternativa correta.

A) Ao implementar um *pipeline* de dois ciclos, o tempo de processamento de uma instrução se reduzirá exatamente em cinquenta por cento, uma vez que cada ciclo executará metade da atividade.

B) Um exemplo de técnica de *pipeline* se chama mestre e escravo, na qual um ciclo de *pipeline*, considerado o ciclo principal, coordena a execução das atividades, dividindo o trabalho entre os demais.

C) Um *pipeline* é chamado de superescalar quando a sua velocidade de processamento atinge o limite de 1000 instruções por milissegundo.

D) O *pipeline* é uma técnica que existe nos processadores desde o computador Tretch IBM 7030. Todas as atuais versões de processadores têm *pipeline* e, quanto mais moderno o processador, maior o seu número de *pipelines*.

E) O uso de *pipeline* é recomendado apenas para aplicações matemáticas, uma vez que para as demais operações usar *pipeline* pode ocasionar instabilidade de dados e consumo.

3) **O conceito básico que norteia a utilização de um *pipeline* é a paralelização de atividades, visando a explorar ao máximo os recursos computacionais, não deixando o processador ficar ocioso. Entretanto, *pipeline* é um conceito que se aplica dentro e fora da área de Tecnologia da Informação (TI), tendo algumas características fundamentais onde for aplicado.**

Analise as alternativas a seguir e assinale a que apresenta duas principais premissas básicas de todo o processo de *pipeline*.

A) Dividir o processo em duas partes e executá-las paralelamente.

B) Dividir o processo em etapas dependentes e interconectadas.

C) Dividir o processo de acordo com o número de núcleos do processador.

D) Dividir o processo em etapas independentes e iniciar uma etapa sem que a outra tenha sido concluída.

E) Dividir o processo em etapas independentes, não iniciando uma etapa antes da conclusão da outra.

4) **Diversos tipos de *pipeline* podem ser implementados pelo processador para executar tarefas. Independentemente do tipo, todos têm um objetivo em comum: proporcionar maior agilidade e eficiência no processamento, reduzindo a ociosidade.**

Assinale a alternativa que contém um tipo de *pipeline* e sua definição correta.

A) *Pipeline* aritmético: utilizado para o processamento de operações aritméticas, como análise sintática de textos.

B) *Pipeline* de instruções: utilizado para realização de cálculos matemáticos.

C) *Pipeline* de controle de fluxo: se divide em cíclico e acíclico.

D) *Pipeline* de funcionalidade: se divide em multifuncional e unifuncional.

E) *Pipeline* cíclico: ocorre quando o primeiro ciclo do *pipeline* se comunica com o último.

5) **Embora o uso de *pipeline* proporcione um ganho de *performance* significativo, quando corretamente utilizado, por ser uma técnica de paralelização, alguns cuidados precisam ser tomados, uma vez que os *pipelines* podem causar alguns problemas no processamento de instruções complexas que tenham interdependência entre as partes que estão sendo processadas por cada ciclo do *pipeline*.**

Sobre os problemas que podem ser causados por *pipeline*, assinale a alternativa correta.

A) Conflito de dados, ocasionado quando o *pipeline* precisa acessar o banco de dados de uma aplicação e não tem permissão para esse acesso.

B) Conflito de memória: ocorre quando a memória do computador não consegue comportar a execução simultânea dos *pipelines*, ocasionando um erro do tipo *segmentation fault*.

C) Conflito de recursos: acontece quando dois ciclos diferentes do *pipeline* precisam acessar um determinado recurso para realizar sua atividade; contudo, esse recurso não comporta acesso simultâneo.

D) Conflito de controle: quando o *pipeline* implementa a arquitetura mestre e escravo e o ciclo mestre responsável pelo controle dos demais está ainda efetuando seu processamento, não conseguindo alocar as tarefas para os demais.

E) Conflito de criptografia: ocorre quando apenas um *pipeline* tem a chave de decriptografia dos dados e todos os *pipelines* precisam acessar esses dados simultaneamente.

NA PRÁTICA

Pipeline é uma tecnologia que foi incluída nos processadores para melhorar o seu desempenho. Pelo *pipeline* é possível realizar uma atividade maior, dividindo-a em atividades menores que, por serem realizadas de forma paralela, podem trazer um ganho de *performance* ao computador. O mesmo conceito de *pipeline* pode ser aplicado via *software*, desde que, ao efetuar a programação desse *software*, o programador se preocupe com a divisão do processamento da

solução algorítmica em etapas que possam ser executadas separadamente por cada núcleo de processador.

Neste Na Prática, você vai ver como funciona um algoritmo de ordenação de vetores implementado utilizando *pipeline*.

ORDENAÇÃO DE VETORES COM PIPELINE

A ordenação de vetores é um caso clássico de utilização de *pipeline*. Diversos são os tipos de ordenação de vetores, sendo a diferença entre eles uma melhor ou pior *performance* e uma maior ou menor dificuldade de programação. Neste exemplo, você vai trabalhar com o algoritmo *bubble sort*.

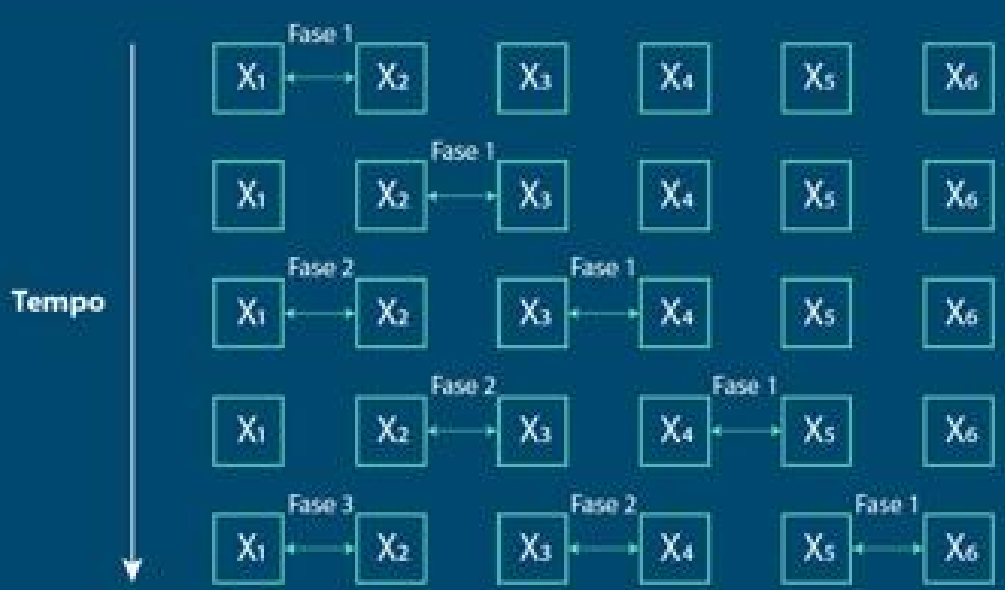
O algoritmo *bubble sort*, quando implementado de forma sequencial, tem seu funcionamento baseado em localizar o primeiro elemento do vetor e, a partir dele, realizar a comparação dele com todos os demais, sendo que, sempre que localiza um elemento com valor maior do que o que está sendo comparado, é feita a troca de posições entre eles. Nesse caso, para garantir a ordenação de um vetor de 100 posições, seriam necessárias 10.000 tentativas, uma vez que o número de tentativas é o tamanho do vetor ao quadrado. Se cada tentativa for um ciclo de relógio, então seriam necessários 10.000 ciclos de relógio.

A figura demonstra uma parte do *bubble sort* sequencial. Note que o processo iniciou com a comparação do primeiro número do vetor com o segundo, e assim sucessivamente, até comparar o penúltimo com o último. Esse ciclo se repete até que todos os elementos sejam verificados e ordenados. No caso de vetor de cinco posições, podem acontecer até 25 comparações e, por serem sequenciais, 25 ciclos de relógio.

1ª passagem do *bubble sort*

Agora, veja um exemplo de funcionamento do algoritmo *bubble sort* utilizando *pipeline*.

Nesta figura, é possível visualizar a simulação de cinco ciclos de relógio, na ordenação de um vetor de seis posições. É possível notar que, inicialmente, o processo tem apenas o ciclo 1 (fase 1) do *pipeline*, trabalhando na comparação de x1 com x2, e depois de x2 com x3. Quando o ciclo 1 está fazendo a comparação de x3 com x4, um ciclo 2 recomeça as comparações do início do vetor comparando x1 com x2 novamente e, quando o ciclo 2 atinge a comparação de x3 com x4, um ciclo 3 retorna a comparação para o início do vetor, comparando x1 com x2. Esse processo se repete até que todos os ciclos disponíveis tenham sido alocados ou até que o vetor tenha sido completamente ordenado.



CONCLUSÃO

Não é possível estimar exatamente o ganho de *performance*, porque está muito relacionado com o desempenho da máquina. Contudo, sabe-se que a versão paralela do algoritmo *bubble sort* tem complexidade O(n), enquanto a versão sequencial tem complexidade O(n2), o que, em se tratando de algoritmos, é considerado uma complexidade elevada.

SAIBA MAIS

Para ampliar o seu conhecimento a respeito desse assunto, veja abaixo as sugestões do professor:

Organização de computadores

Neste vídeo, você vai ver um apanhado geral sobre o conceito de *pipeline*, assim como o seu funcionamento.

Conteúdo interativo disponível na plataforma de ensino!

Pipeline

Este artigo apresenta algumas informações sobre os principais problemas de conflitos que acontecem com *pipeline*.

Conteúdo interativo disponível na plataforma de ensino!

Arquiteturas paralelas e distribuídas

O material a seguir apresenta um ótimo resumo sobre *pipeline* por meio de desenhos e explicações didáticas.

Conteúdo interativo disponível na plataforma de ensino!